

Formal Engineering of Reliable Software

Natasha Sharygina
Carnegie Mellon University

*LASER 2004 school
Tutorial, Lecture 1*

Project Goals

To Build Reliable and Robust Software Systems

by

- 1) Integrating Systems Engineering with Formal Verification techniques
- 2) Enabling Model Checking of Realistic Software Systems

Outline

Lecture 1, part 1

- Motivation
- Model Checking

Lecture 1, part 2

- State/Event-based software model checking

Lecture 2

- Component Substitutability

Outline

Lecture 1, part 1

- Motivation
- Model Checking

Lecture 1, part 2

- State/Event-based software model checking

Lecture 2

- Component Substitutability

Motivation

Goal: Build reliable computer systems

- Secure and safe execution
- Predictable designs (no unexpected behaviors)

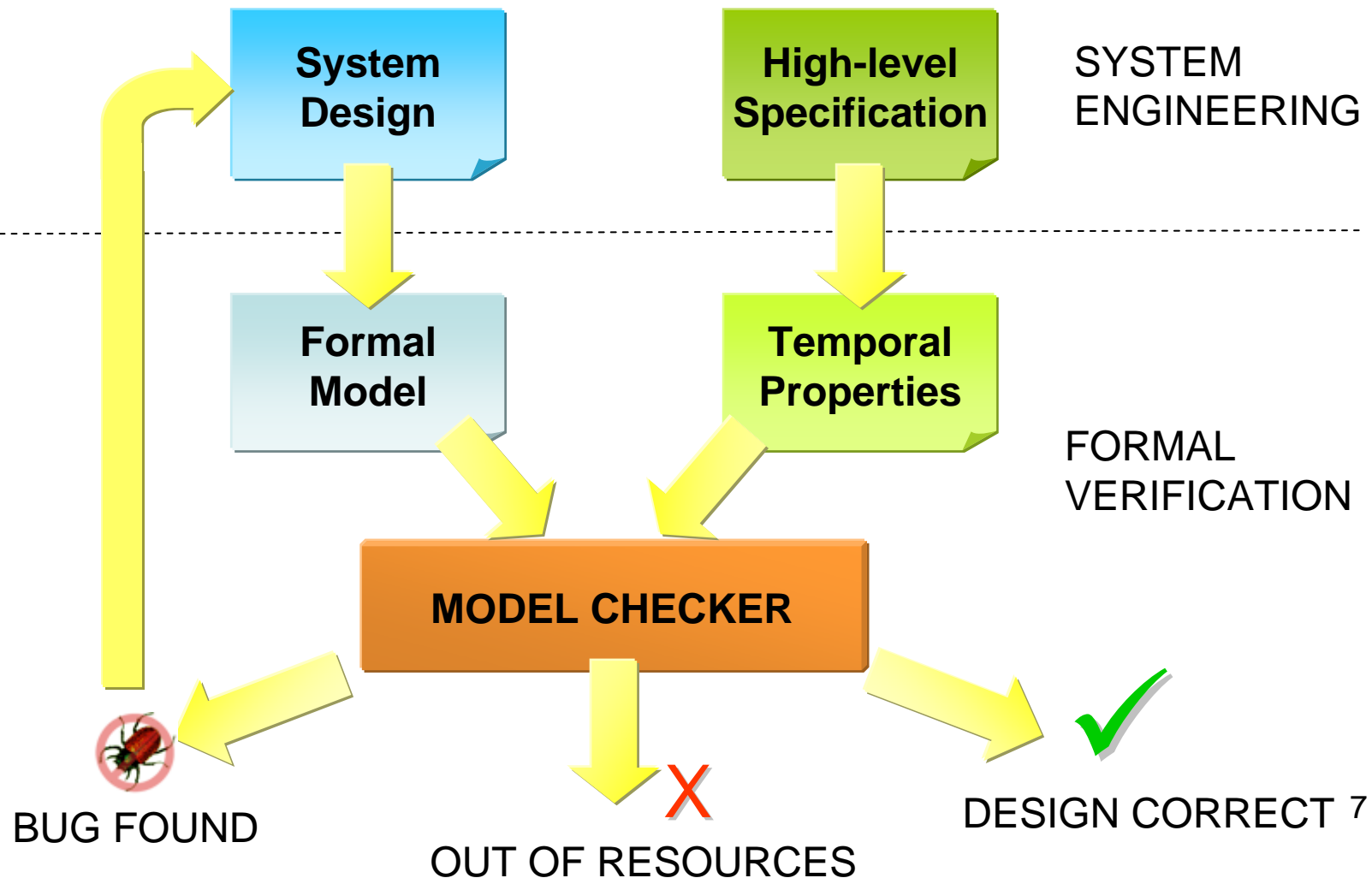
Applications: embedded systems in
avionics, space, robotics,
electro-mechanical engineering, etc.

Motivation

Approach: Integrate Validation and Verification with Systems Engineering

- Reasoning about system designs during their construction
- Design for verification

ComFoRT: Component Formal Reasoning Framework



CCL Modeling Language

A CCL system is a parallel composition of individual sequential programs,

$$P = p_1 \parallel \dots \parallel p_n,$$

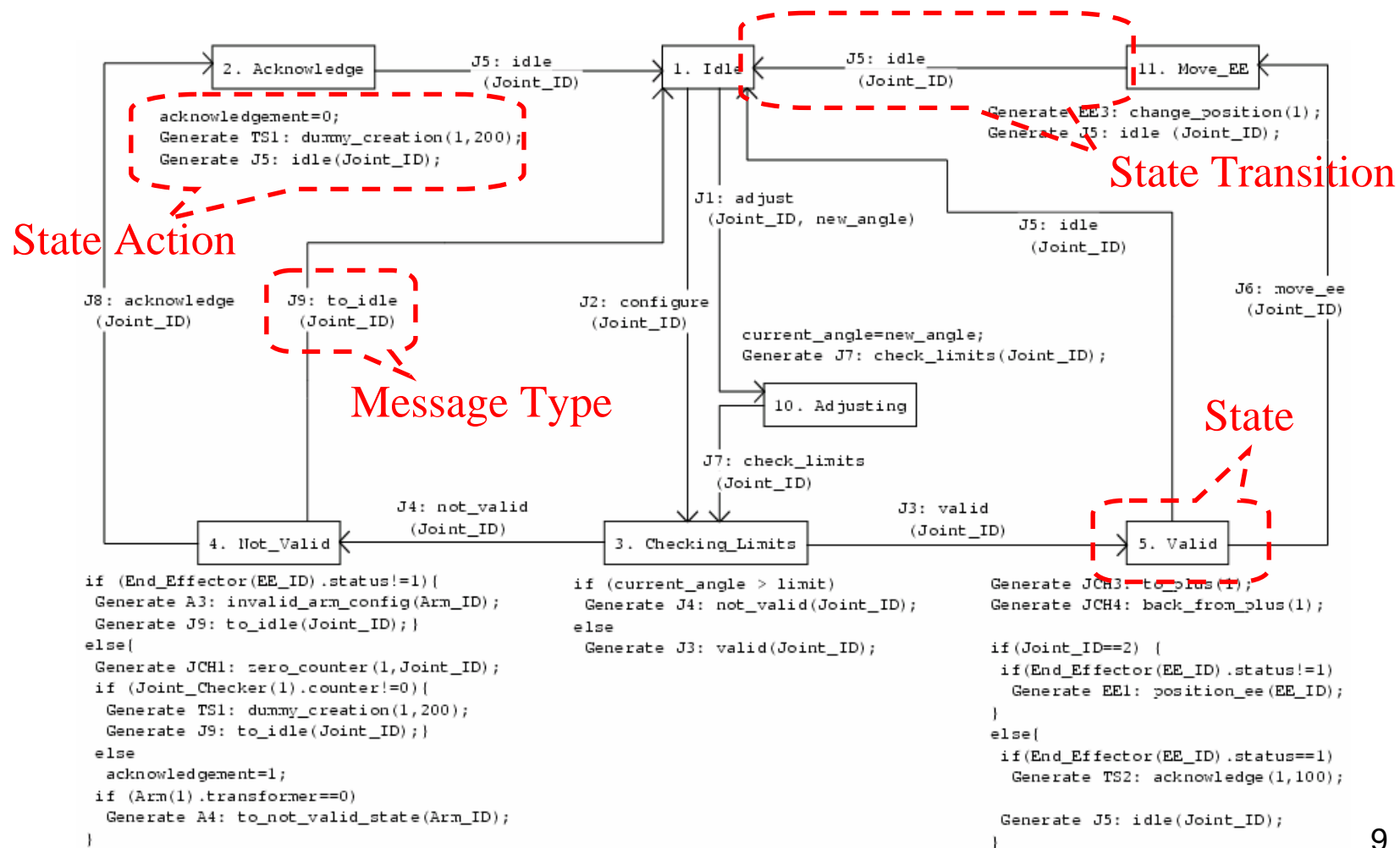
Sample commands of CCL programs:

Assignments: **$x := \text{exp} \mid x := \text{any}\{\text{exp}_1, \dots, \text{exp}_n\}$**

Communication: : **Generate $e_i(ID, \text{exp})$** - Event generation
 Receive $e_i(ID, x)$ - Event consumption

Compounds: ***if then else, while do od, switch***

Sample CCL state model



Outline

- Motivation
- **Model Checking**
- State/Event-based software model checking
- Component Substitutability

Temporal Logic Model Checking

- Systems are modeled by **finite state machines**.
- **Properties** are written in **propositional temporal logic**.
- Verification procedure is an **exhaustive search of the state space** of the design.
- **Diagnostic counterexamples**

What is Model Checking?

**Does model M satisfy a property P ?
(written $M \models P$)**

What is “ M ”?

What is “ P ”?

What is “satisfy”?

What is “M”?

States: valuations to all variables

Initial states: subset of states

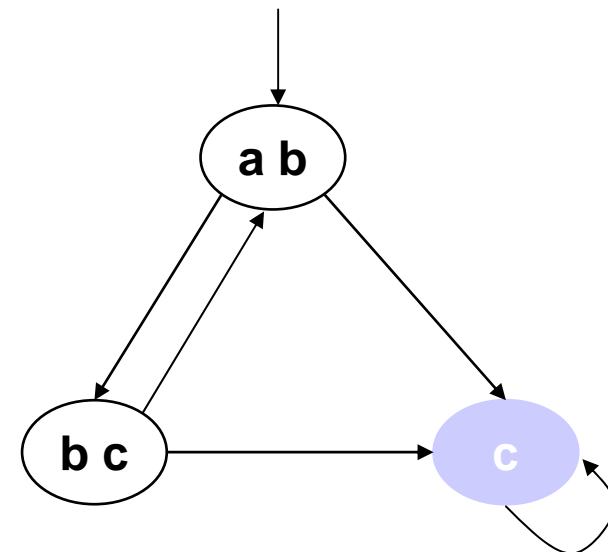
Arcs: transitions between states

Atomic Propositions:

e.g. $x = 5$, $y = \text{true}$

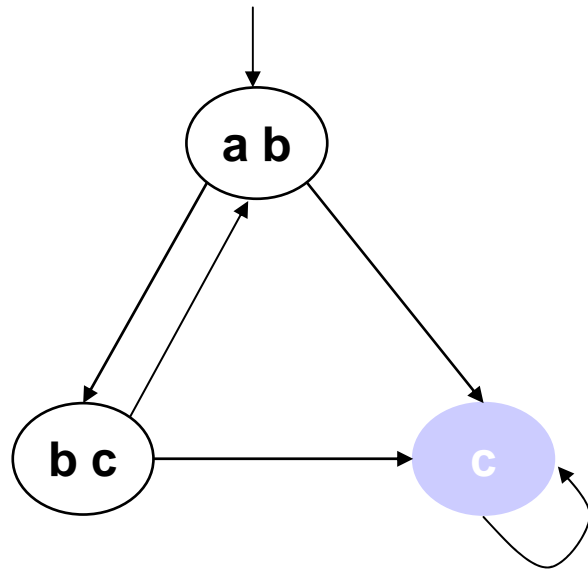
Observation (color):

Valuation to all atomic propositions

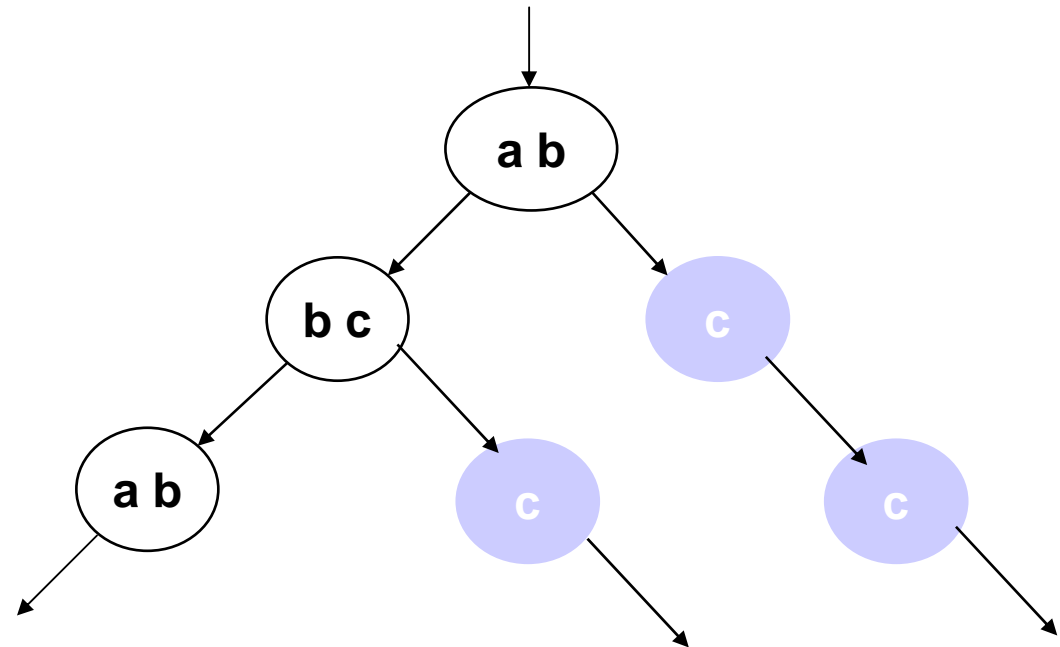


State Transition Graph or Kripke Model

Model of Computation



State Transition Graph



Infinite Computation Tree

Unwind State Graph to obtain Infinite Tree.

A *trace* is an infinite sequence of states.

What is “P”?

Syntax: What are the property formulas?

Semantics: What does it mean for model **M** to satisfy formula **P**?

Formulas:

- Atomic propositions: properties of states
- (Linear) Temporal Logic Specifications: properties of traces.

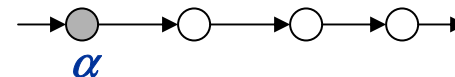
Specification (Property)

Examples: **Safety** (mutual exclusion): no two processes can be at the critical section at the same time

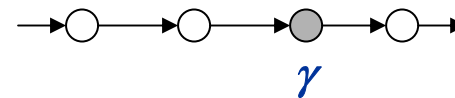
Liveness (absence of starvation): every request will be eventually granted

Linear Time Logic (LTL) [Pnueli 77]: logic of temporal sequences.

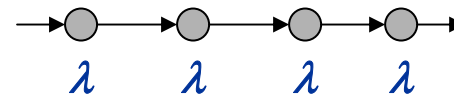
• **next (α):** α holds in the next state



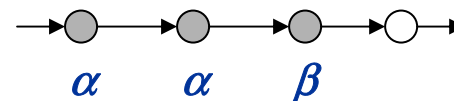
• **eventually(γ):** γ holds eventually



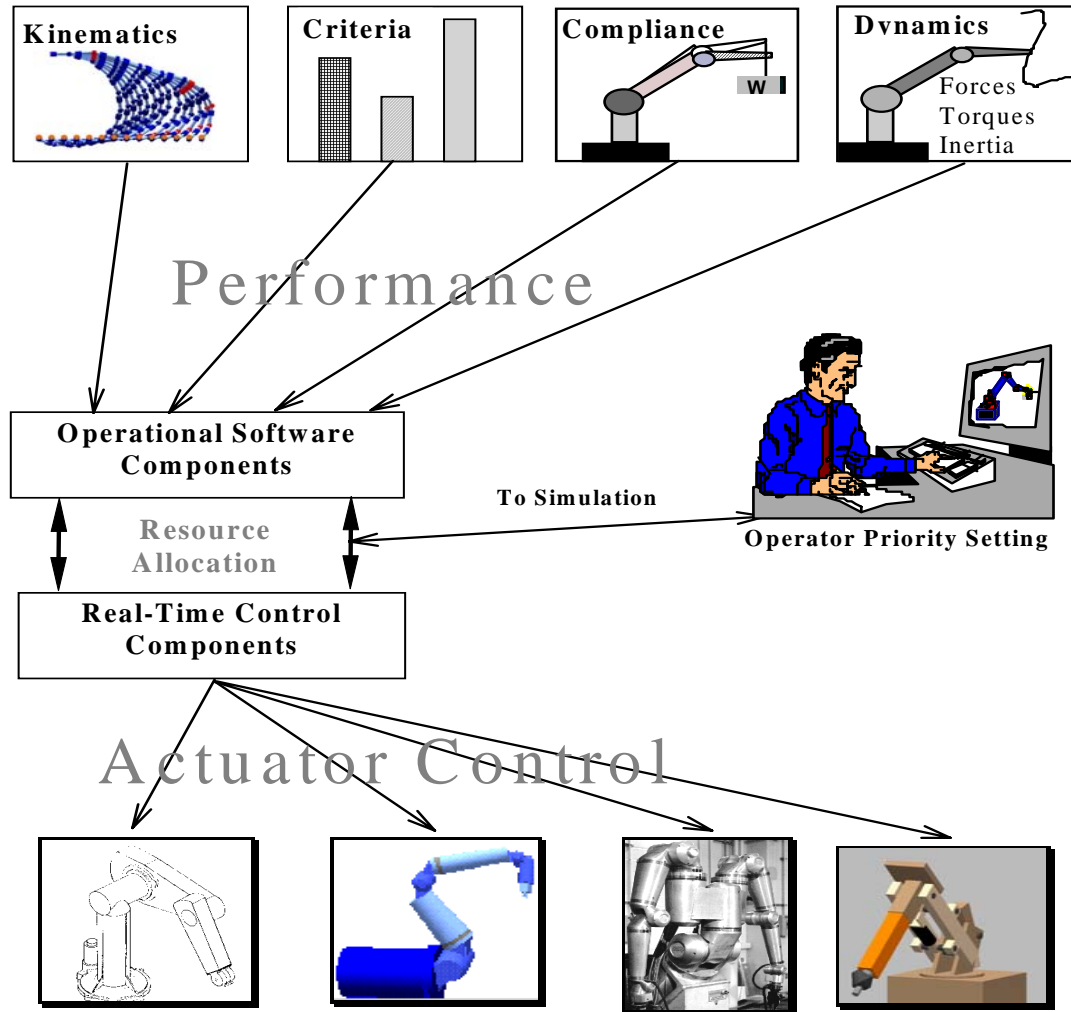
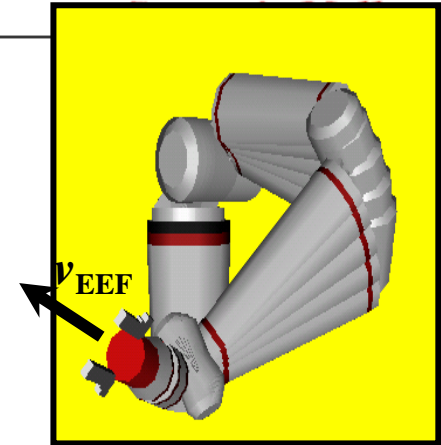
• **always(λ):** λ holds from now on

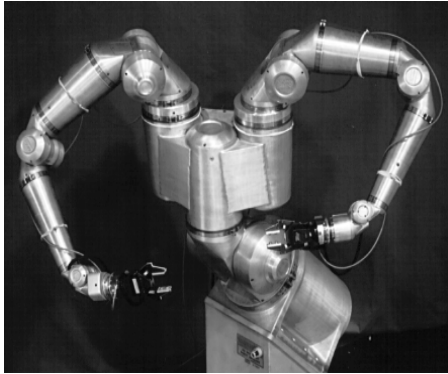


• **α until β :** α holds until β holds

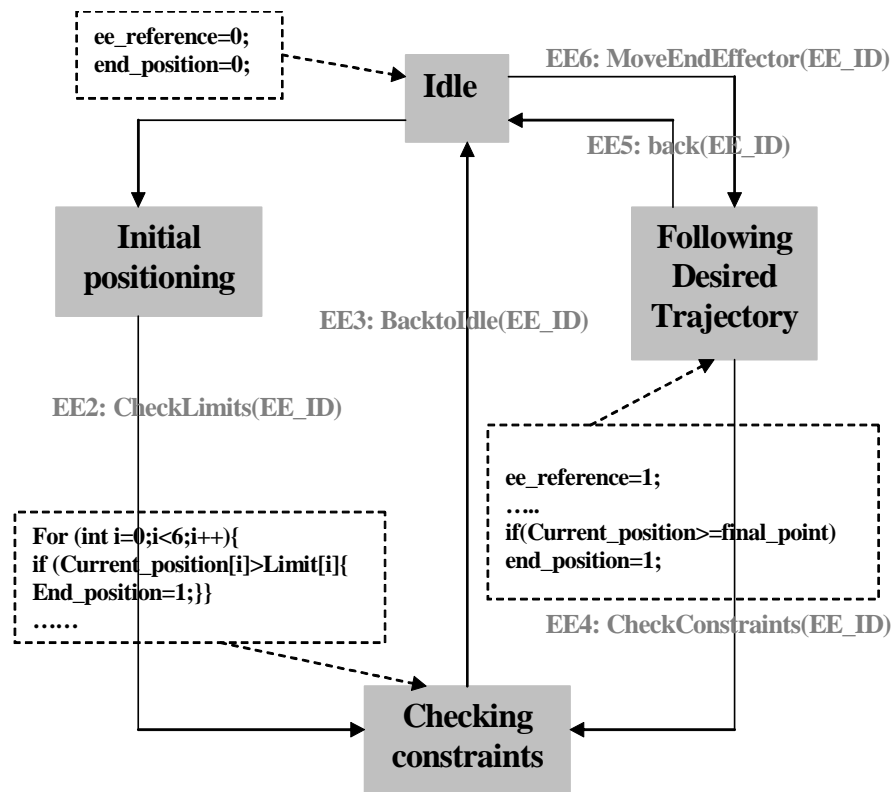


NASA Robot Controller System

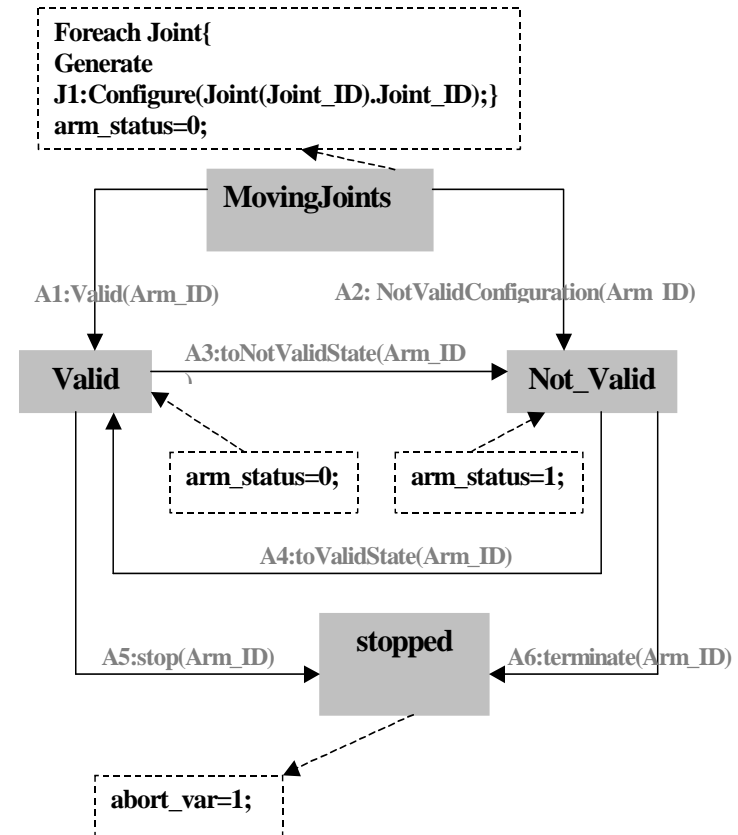




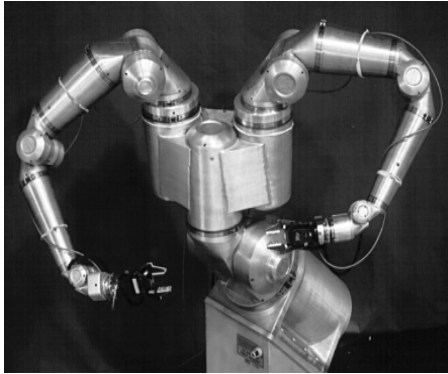
Modeling of the NASA Robot Controller System



EndEffector



Arm



Examples of the Robot Control Properties

- **Safety Operation:** If the *EndEffector* reaches an undesired position, then the program terminates prior to a new move of the *EndEffector*

AfterAlwaysUntil(undesired_position =1,ee_reference=1,abort_var=1)

- **Configuration Validity Check:**
If an instance of *EndEffector* is in the “FollowingDesiredTrajectory” state, then the instance of the corresponding *Arm* class is in the “Valid” state

Always((ee_reference=1) ->(arm_status=1))

- **Control Termination:** Eventually the robot control terminates

EventuallyAlways(abort_var=1)

What is “satisfy”?

M satisfies P if *all* the reachable states satisfy P

Different Algorithms to check if $M \models P$.

- Explicit State Space Exploration

For example: Invariant checking Algorithm.

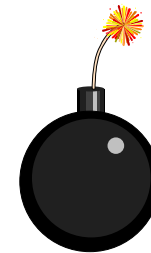
1. Start at the initial states and explore the states of M using DFS or BFS.
2. In any state, if P is violated then print an “error trace”.
3. If all reachable states have been visited then say “yes”.

State Space Explosion

Problem:

Size of the state graph can be exponential in size of the program (both in the number of the program *variables* and the number of program *components*)

$$M = M_1 \parallel \dots \parallel M_n$$



If each M_i has just 2 local states, potentially 2^n global states

Research Directions: State space reduction

State Space Explosion

Principal Approaches to State Space Reduction:

- *Abstraction*
(elimination of details irrelevant to verification of a property)
- *Compositional reasoning*
(reasoning about parts of the system)
- *Symbolic Verification*
(BDDs represent state transition diagrams more efficiently)
- *Partial Order Reduction*
(reduction of number of states that must be enumerated)
- *Other*
(symmetry, cone of influence reduction,)

Systems engineering and model checking

Principal Approach

- *Component-based system design*
- *Compositional reasoning*
(reasoning about parts of the system)

Components

Compositional reasoning reduces reasoning about entire system to reasoning about individual parts

- Decompose the model: $M = M1 \parallel M2$
- Partition global properties into local properties: $P = P1 \wedge P2$
- Show that $M1 \models P1$ and $M2 \models P2$

Component-based design

- *Library of verified components* \rightarrow *predictable designs*